# CIT360 FINAL PROJECT

Jeremy Alkire

I’ve spent much of the past couple of weeks familiarizing myself with Android Studio and mobile application development to gauge what I’m capable of. This, to me, was a prerequisite that ensured that I would not put forward any false promises and would only make a project proposal that I could deliver on.

After careful consideration, I’ve decided that what I want to create is a project tracker. This was inspired by someone I know who does freelance artwork for a multitude of clients in a variety of art styles with varying prices. It’s a lot to keep track of, and a mobile application that allows them to create new project entries and store their work in a database will help them to keep track of the following:

* Project names.
* Client names.
* Character names.
* Prices.
* Art styles.
* Quantity of people in the artwork.
* % progress completed on the art project.
* Any other specifications or notes they want to add.

The Android application will allow them to add new project entries to a database and view projects that have already been added to the database afterwards. It will work in the following manner:

The client side (and view) of the application will be on Android. This will be where the user interface is, and where the user creates new project entries and views old ones.

The main activity will contain two buttons, one titled “View Projects” and one titled “Add Projects”. If they select “Add Projects” they’ll be met with a user interface that looks like a standard input form, composed of Android TextViews, Spinners, and Buttons. Upon submitting their project specifications, the Android application will use Java’s HttpURLConnection class to connect to a servlet hosted on an Apache Tomcat server via the Eclipse IDE, which will compose the server side of the application. In doing so, it will send a POST request to the servlet containing the information that the user wishes to store in the database.

From here, the servlet will accept the Android application’s POST request, parse the sent data, and use it to configure and create a new Project object (an instantiation of a POJO class called “Project”). The servlet will then use the object relational mapping tool, Hibernate, to map the Project object to a corresponding table in the MySQL database, before storing it there. The MySQL database will be hosted via XAMPP.

Upon storing the project information in the database, the servlet will then use hibernate to query the database for all existing entries in the Projects table before using it to populate a List of Project objects. After populating the list, the servlet will iterate through this list and convert each object to a JSON string, only to append it to the message string that it will ultimately send back to the Android application via HttpURLConnection.

Android’s AddProjects Activity will take this single string of concatenated JSON data, split it via the delimiter character to populate an array, before sending it over to the ViewProjects Activity. The ViewProjects Activity will then create an empty ArrayList of type Project. To populate this ArrayList, the ViewProjects Activity will iterate through the array of JSON strings sent to it by the AddProjects activity and convert each element to a Project object before adding it to the ArrayList. To do this, the program will call a convertToJava() method that will use a HashMap to parse the JSON string data sent to it from the Array of JSON strings.

From here, the ViewProjects Activity will iterate through the ArrayList of Project objects and an ArrayList of TextViews. While iterating through these lists, the program will call the getter() methods of each Project object in the Project ArrayList and display them in each TextView by calling the setText() method. Ultimately, this will display the contents of the SQL database back to the user via the Android application.

In the background of all of this, the servlet itself will run a separate thread that simply counts the seconds elapsed during the servlet’s lifetime and prints it to the console for server-side logging information.

In total, this project will utilize:

* Hibernate
* Android (Emulator on Android Studio)
* QuickConnectFamily JSON Library
* Threads
* Servlets
* Collections (ArrayList, List, and HashMap)
* HttpURLConnection
* Eventually JUnit for testing.
* Use Case Diagrams, Use Case Documents, Sequence Diagrams, and State Diagrams

Regrettably, I am not familiar enough with Android to consider utilizing some sort of Application Controller Pattern or Model View Control architecture.

A design plan is below:

